I3000 User's Guide

Revision: V1.20 Date: August 22, 2014

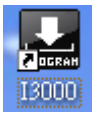
www.holtek.com

Table of Contents

1 I3000 Usage 3
1.1 Start Interface 3
1.2 Main Interface 7
1.3 Area Functions in the Main Interface 8
1.4 Functional Introduction 8
1.4.1 Erase 8
1.4.2 Verify 13
1.4.3 Program 14
1.4.4 Blank Check 15
1.4.5 Partial Lock 16
1.4.6 Partial Program 17
1.4.7 Switch to User Program 18
1.4.8 MCU Start 18
1.4.9 Show Checksum 19
1.4.10 Bootloader Version 19
1.5 Using the I3000 for Program Operation 20
2 Software Development Kit – SDK 23
2.1 Operating Introduction 23
2.2 Operating Flowchart 23
2.3 SDK API Introduction 24
LoadFile 24
LoadProgdata 25
SaveFile 26
GetChksum 27
BlankCheck 28
Program 29
PartialProgram 30
EraseByAddr 31
EraseByPage 32
VerifyByAddr 33
VerifyByPage 34
LockByAddr 35
LockByPage 36
LockFromOption 37
LockAll 38
GetTransProgress 38
GetBootloaderVer 39
ConnectToBootloader 39
DisConnectBootloader 39
SwitchToUserProgram 40
ExecuteProgramFrom 40
GetMCUInfo 41
i3kSys 42

1 I3000 Usage

1.1 Start Interface

After the I3000 installation is finished, double-click the 

icon on the desktop, the following interface will then appear:

Figure 1

"USB Connect" indicates whether the software has connected to the Bootloader, "Download Mode" indicates if it is ready for program operations. means yes, means no. Figure 1 indicates that it has been connected to the Bootloader and normal program operations are allowed, while Figure 2 indicates no.

Figure 2

Click the <Setting> button to select the data type. Regarding the white check boxes, means has been selected and means not selected.

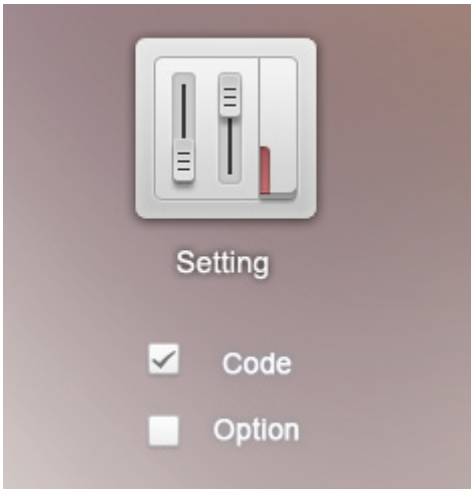


Figure 3

Click the <Help> button to view the software information, clicking the following link will open this User's Guide.

Figure 4

Move the mouse to the button next to "Exit". When the button turns red as the following shows, a single click on the button will exit the software.

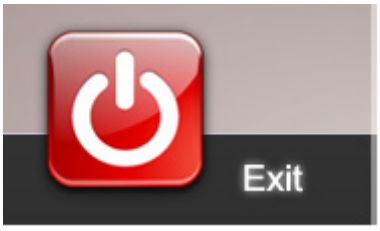


Figure 5

Click the <Open> button, an open file dialogue box appears. Only those files with an mtp suffix can be selected from any location on the computer.

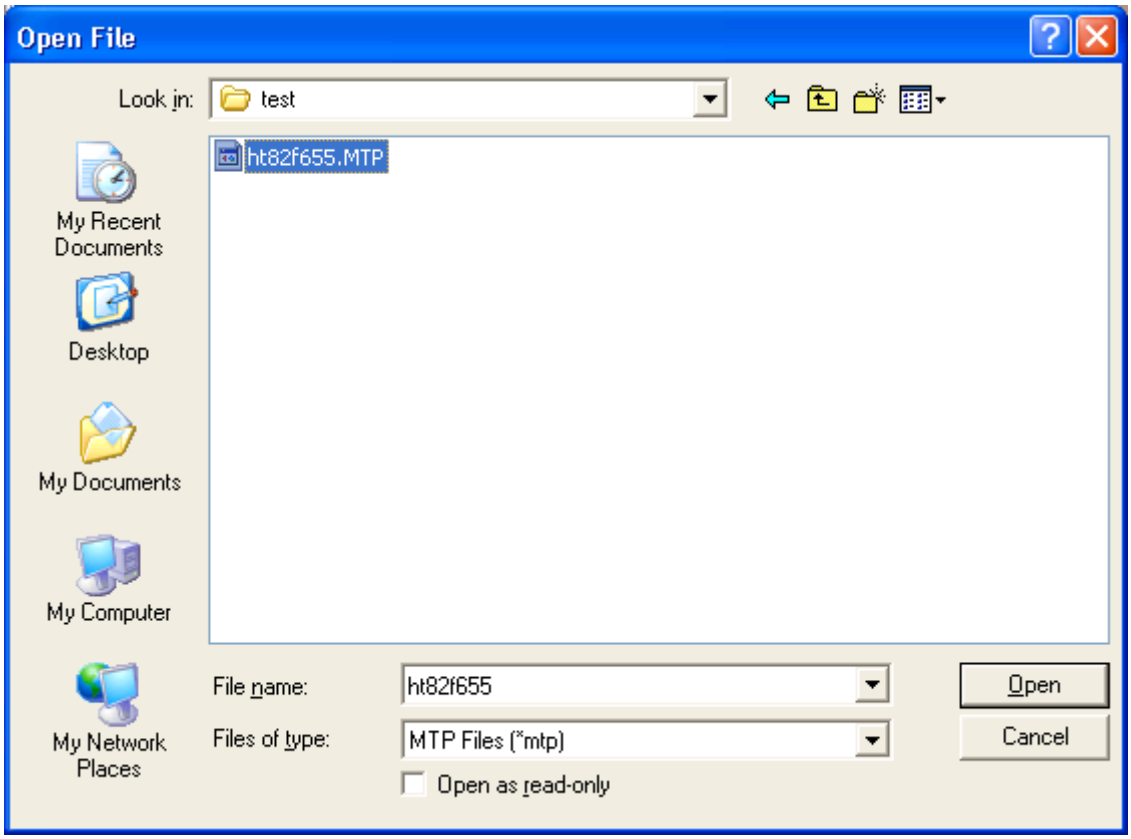


Figure 6

Double-click the required file or single-click the file and click the <Open> button to open the file and enter the main interface, as shown in Figure 6.

1.2 Main Interface

The main Interface is mainly composed of the title bar, data area, function button area and status area, as Figure 7 shows.

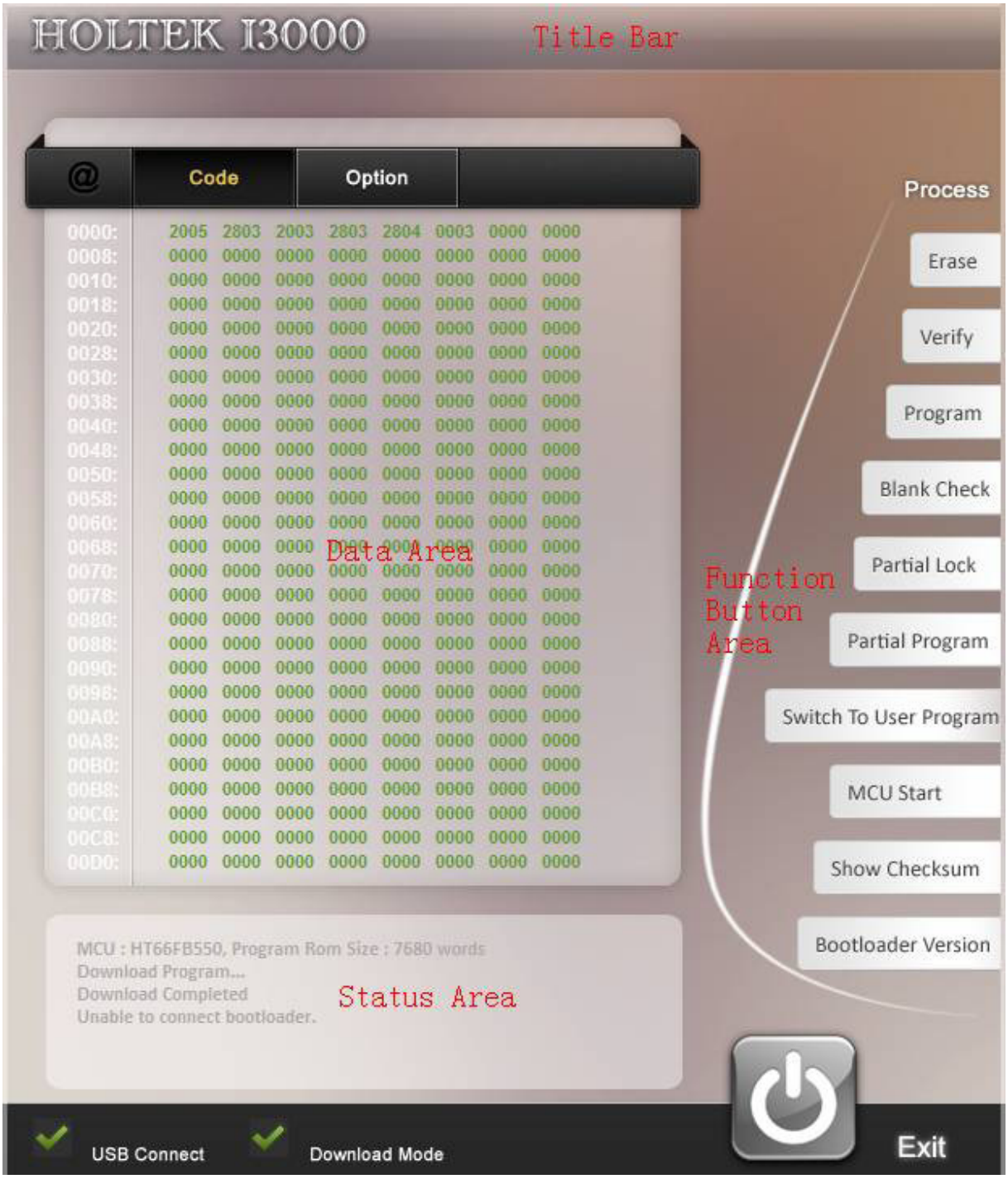


Figure 7

Move the mouse near to the title bar and a tool bar will appear as shown below.

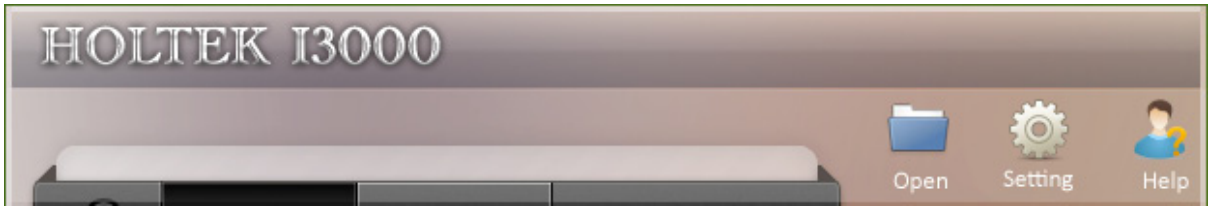


Figure 8

By using this tool bar, users can open new files, reset the data type or view the software information.

1.3 Area Functions in the Main Interface

The data area includes two sections, one for code and one for options. These can be switched between each other with a single click. If not all the contents are shown, move the mouse to the right border, a scroll bar will then appear, with which users can view the contents. The function button area includes all the available operations. The status area is mainly used to display the operation results.

1.4 Functional Introduction

1.4.1 Erase

This instruction erases the code or option information programmed to the IC.

Click the <Erase> button and a dialogue box, as shown in Figure 9, will appear.

Erase Setting
Start Page
Addr:
H
H
End Page
Select All
Unselect All
Click
Cancel
Erase

Figure 9

By inputting a hexadecimal number in the Start Page or End Page text box, users can select the page where this address exists. Press <Click> for a drop-down list to check the selected pages. Users can also select pages directly in the drop-down list. If the selected pages are consecutive, then the corresponding start address and end address will be displayed in the Start Page and End Page text box respectively, as shown in Figure 10.

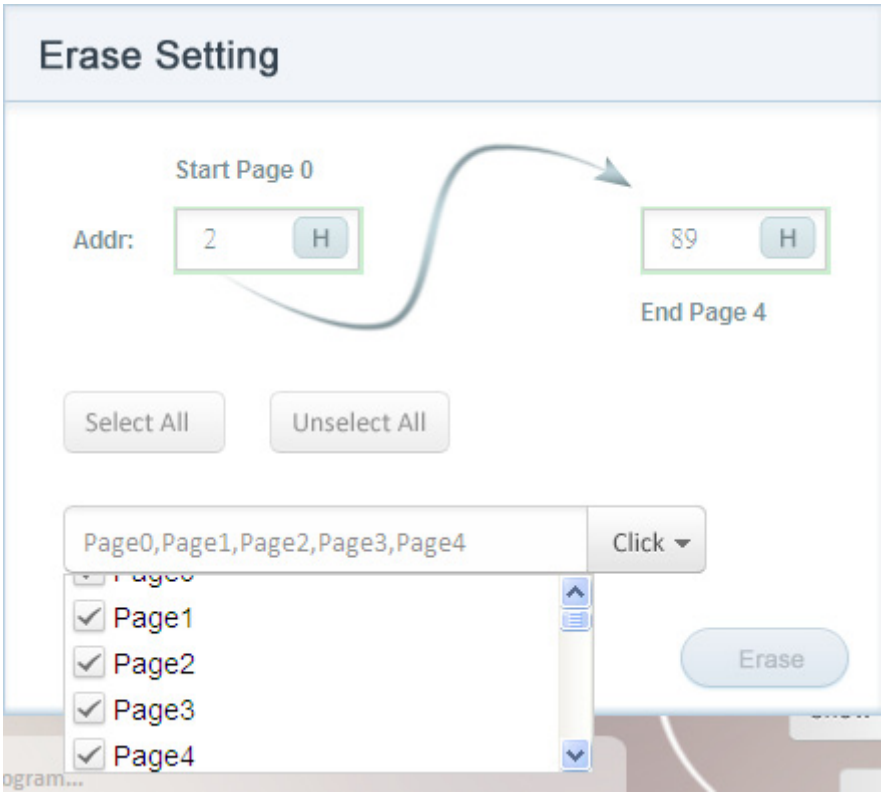


Figure 10

Clicking the <Select All> button will select all pages, clicking the <Unselect All> button will cancel all the selected pages.

Clicking the <Cancel> button will cancel the erase operation.

Clicking the <Erase> button will start the erase operation. If no page is selected or the input address is not valid for a correct page range, a warning as shown below appears.

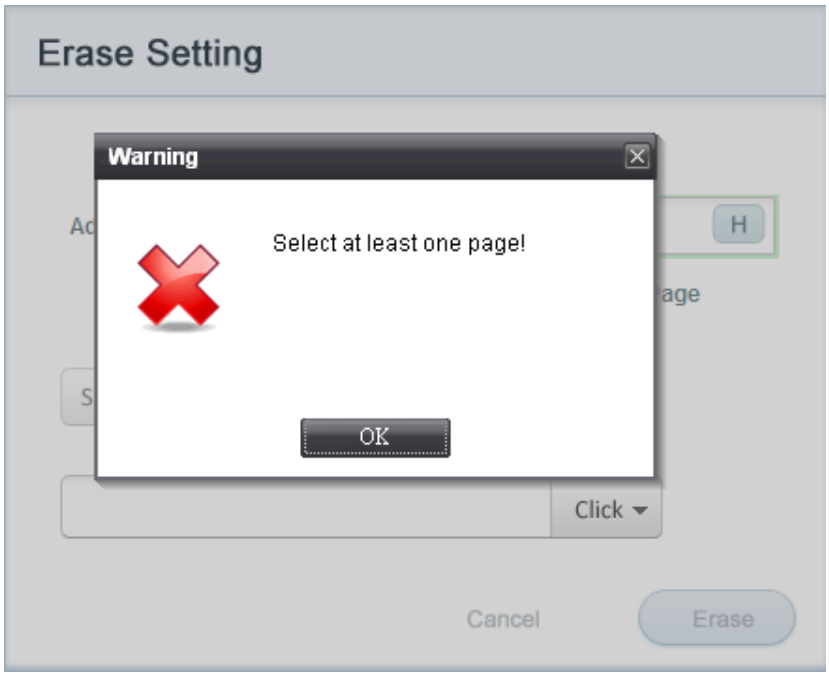


Figure 11

Otherwise a normal erase operation starts and a progress bar appears. If only the code check box in the Setting menu is selected, when the progress bar displays 100%, the status area will display "Erase program OK", and a <Close> button will appear. This is used to close the progress bar as shown in Figure 12.

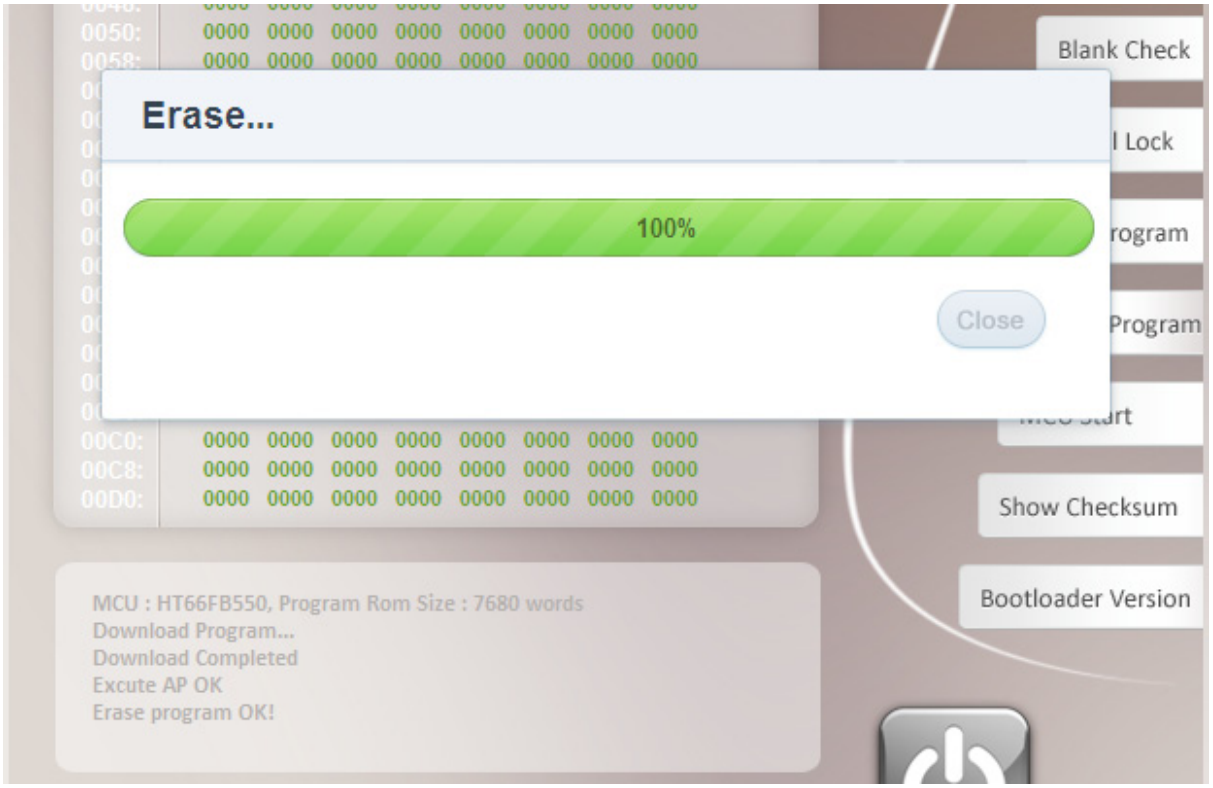

Erase...
100%
Close
Blank Check
Lock
rogram
Program
Show Checksum
Bootloader Version
MCU : HT66FB550, Program Rom Size : 7680 words
Download Program...
Download Completed
Excute AP OK
Erase program OK!

Figure 12

1.4.2 Verify

This instruction verifies whether the code or option information has been programmed to the IC successfully.

Click the <Verify> button and a Verify Setting dialogue box will appear.

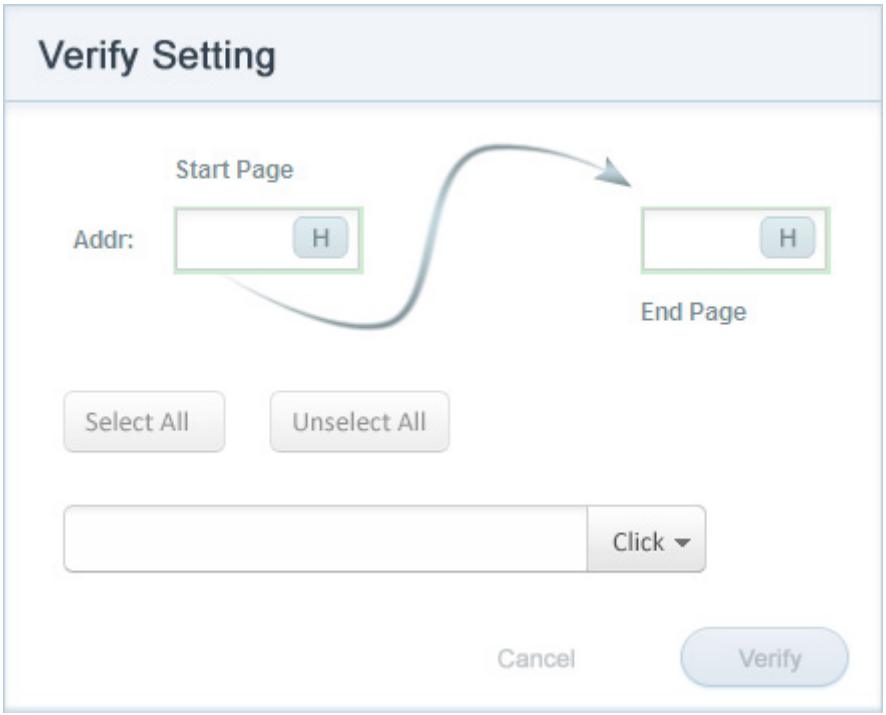


Figure 13

This dialogue box is operated in the same way as the Erase Setting dialogue box.

1.4.3 Program

This instruction programs code or option information to the IC.

Click the <Program> button to start the program operation. This instruction is executed according to the selection in the Setting menu.

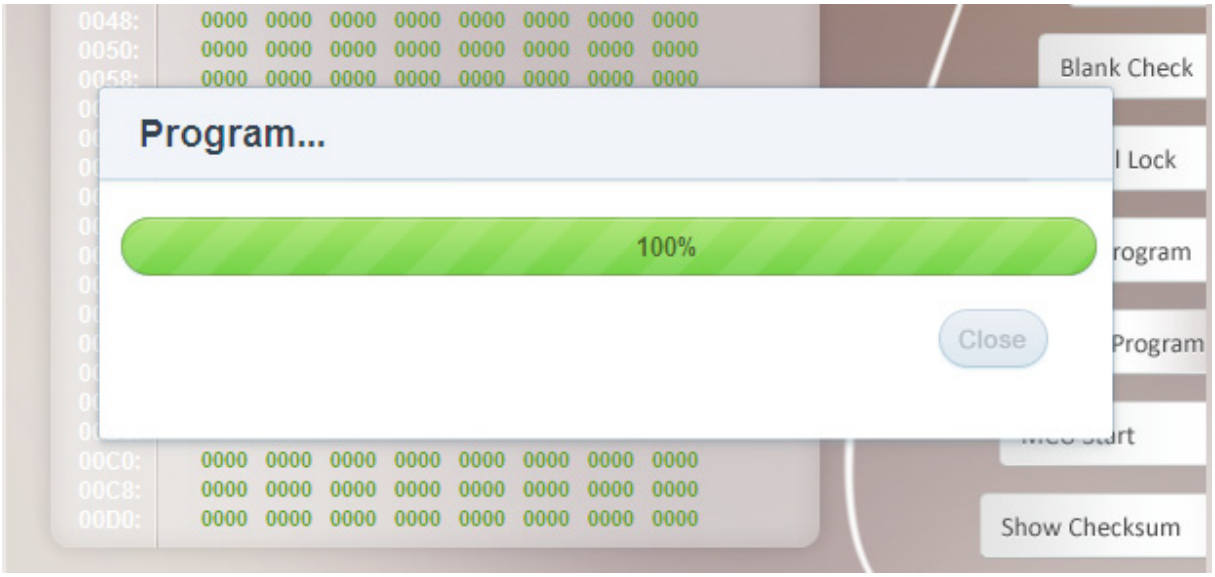


Figure 14

Note: If an error warning appears during the program operation, execute an erase operation and try again.

1.4.4 Blank Check

This instruction checks if the IC has been previously programmed.

Click the <Blank Check> button. If the IC has been programmed, a "Not Blank" error warning will be displayed. A <Close> button will then appear, as shown in Figure 15. Otherwise the <Close> button appears only after the progress bar displays 100%, as shown in Figure 16.

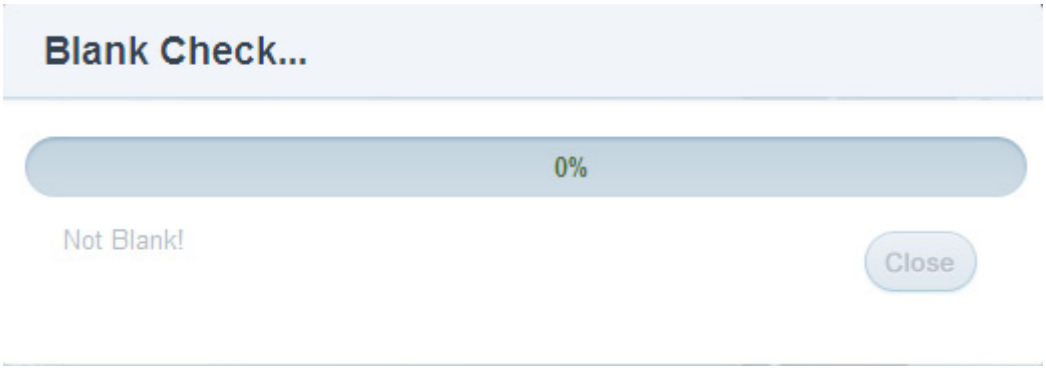


Figure 15

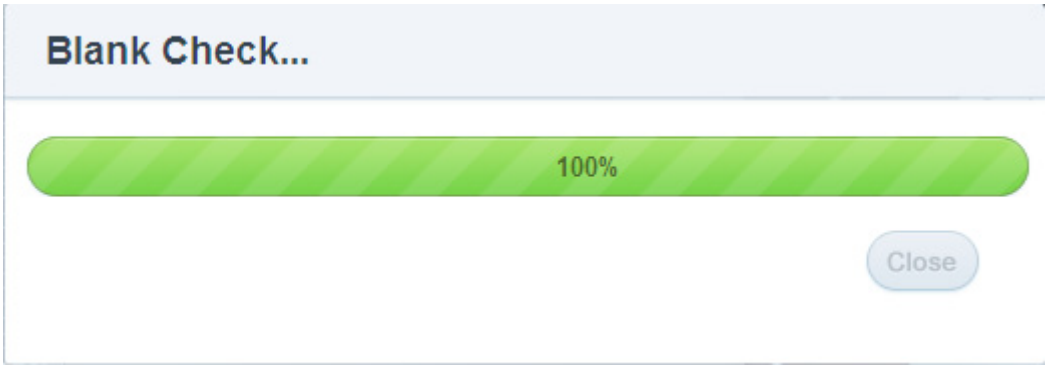


Figure 16

1.4.5 Partial Lock

This instruction locks the selected code section. After being locked, the code cannot be read or erased.

Click the <Partial Lock> button and a dialogue box as shown below appears.

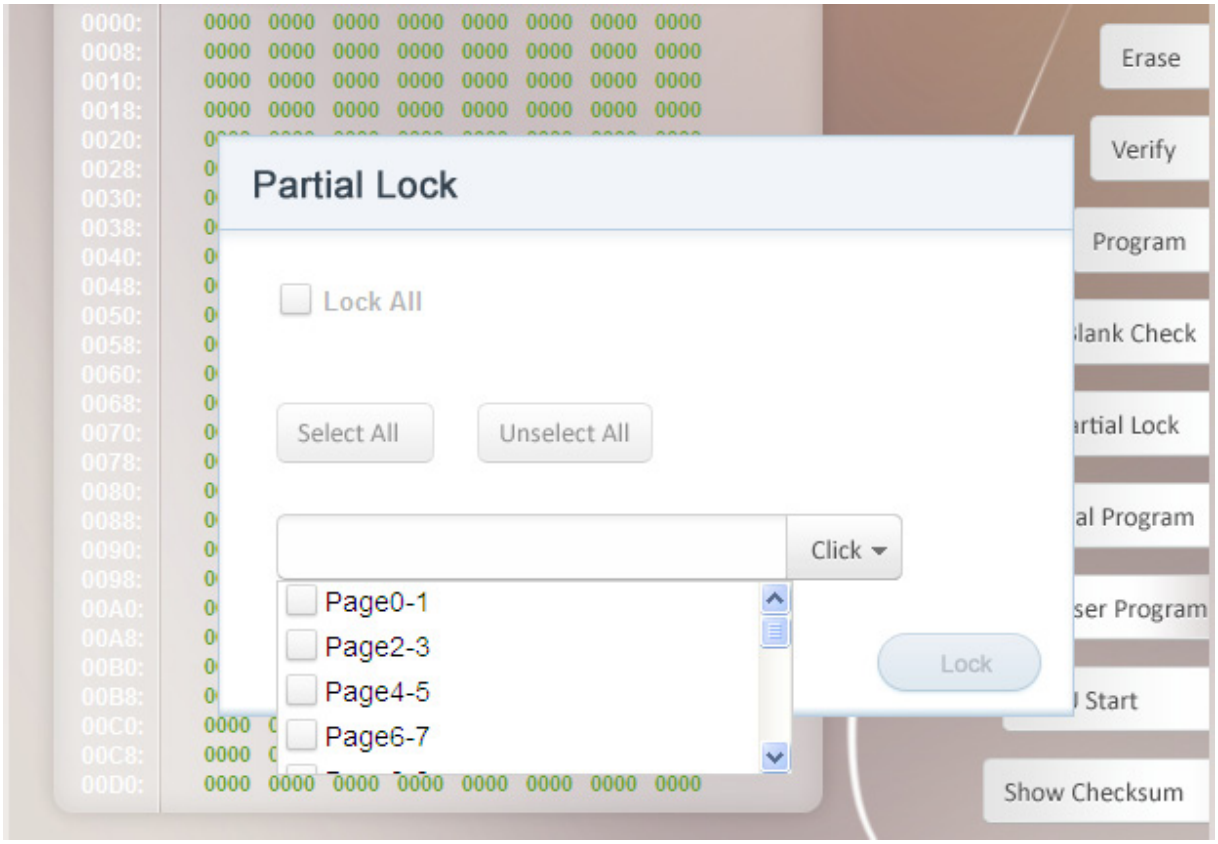


Figure 17

Press <Click> to view the drop-down list. Clicking the white check box will select the corresponding page. Clicking the <Select All> or <Unselect All> button will select all pages or cancel all the selected pages. After selecting the required pages, click the <Lock> button to start the lock operation.

1.4.6 Partial Program

This instruction works in the same way as the Program instruction except that this instruction only programs specified pages.

Click the <Partial Program> button and the following dialogue box appears.

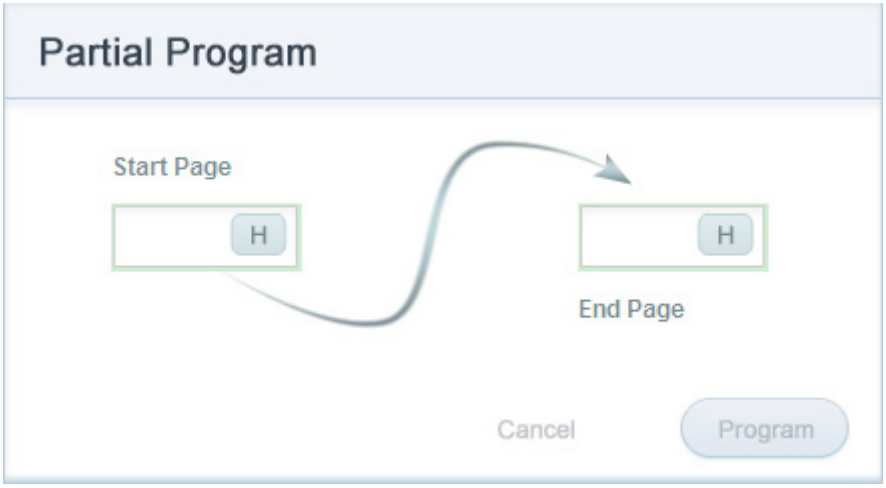


Figure 18

Input a hexadecimal number in the Start Page and End Page text boxes respectively, and ensure that the latter value is bigger than or equal to the former value. Click the <Program> button to program the selected pages.

1.4.7 Switch to User Program

This instruction simulates a USB plug in and plug out action.

Click the <Switch to user program> button and the following dialogue box appears.

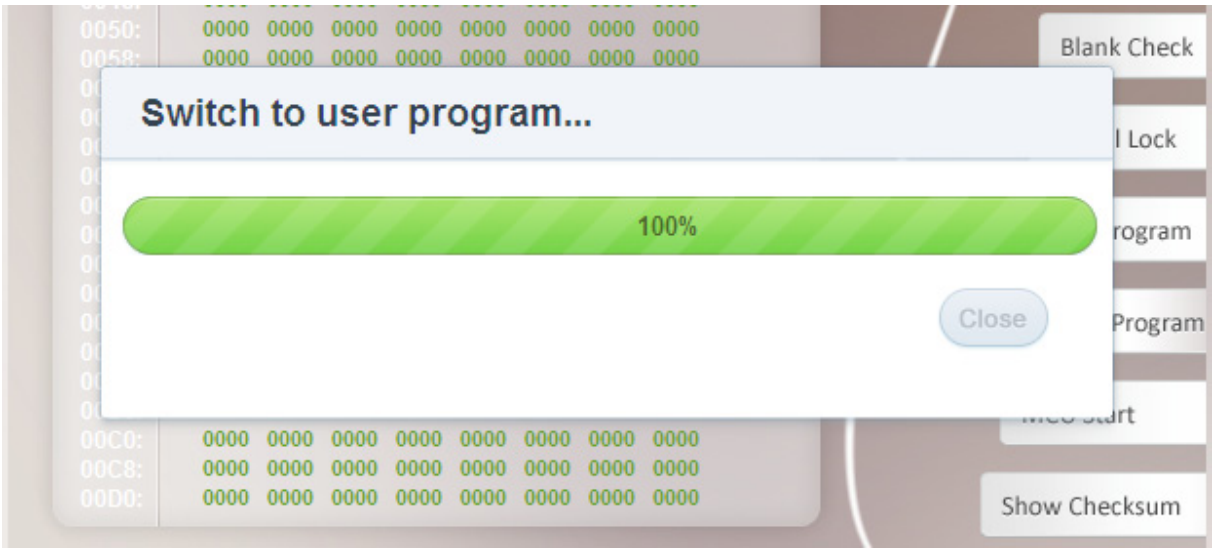


Figure 19

1.4.8 MCU Start

Only after verifying that the IC has been successfully programmed, can this instruction be used to select the MCU start mode.

Click the <MCU Start> button and a dialogue box as shown in Figure 20 appears.

MCU Start
MCU starts at the bootloader
MCU starts at the bootloader then executes the user firmware
MCU starts at the user firmware
Cancel
OK

Figure 20

The functions of the three options shown should be self explanatory from their content.

1.4.9 Show Checksum

Selecting "Code" means to check the Code ROM only.

Selecting "Code+Option" means to check the Code ROM and the Option ROM.

Click the <Show Checksum> button and a dialogue box as shown in Figure 21 appears.

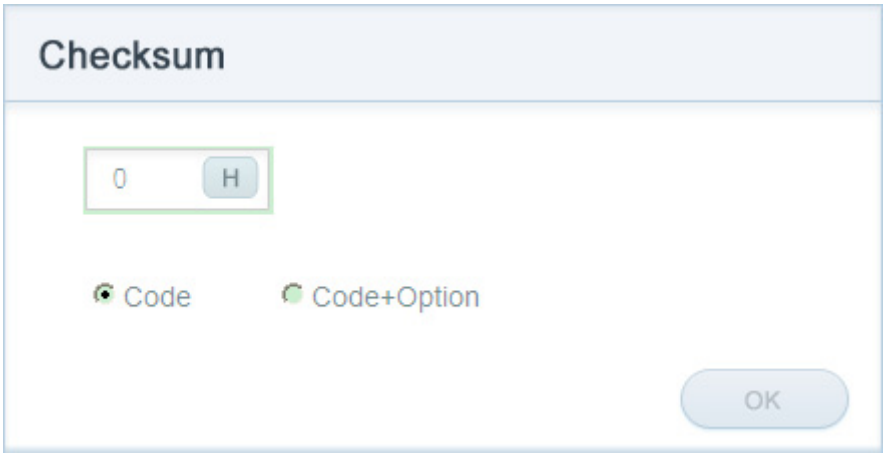


Figure 21

Select the "Code" or the "Code+Option" check box and the text box will display the corresponding Checksum value.

1.4.10 Bootloader Version

This instruction is used to obtain the Bootloader version.

Click the <Bootloader Version> button and the Bootloader version information will be displayed in the status area, as shown in Figure 22.

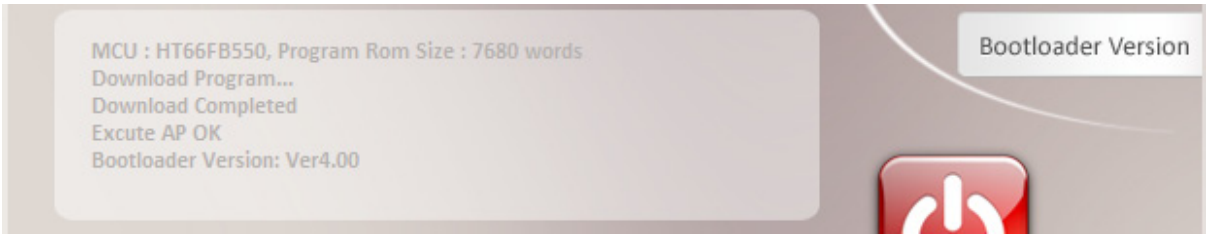


Figure 22

1.5 Using the I3000 for Program Operation

- Step 1

  Click the <Open> button in the I3000 start interface or main interface to open the required MTP file, as shown in Figure 23 and Figure 24.

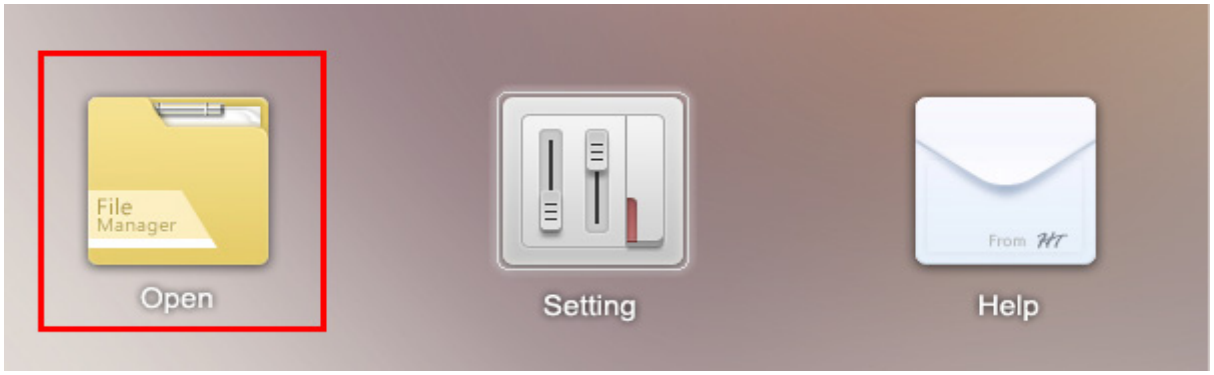


Figure 23

Figure 24

- Step 2

  Ensure that there is a connection to the bootloader. Click the <Program> button to begin program operation, as shown in figure 25. A process bar will then appear, as shown in Figure 26.

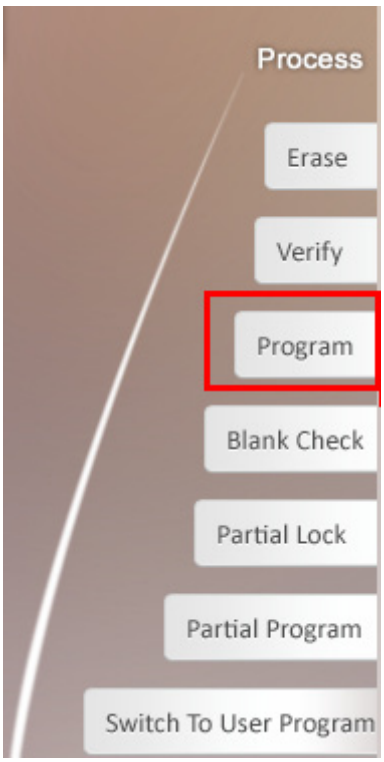


Figure 25

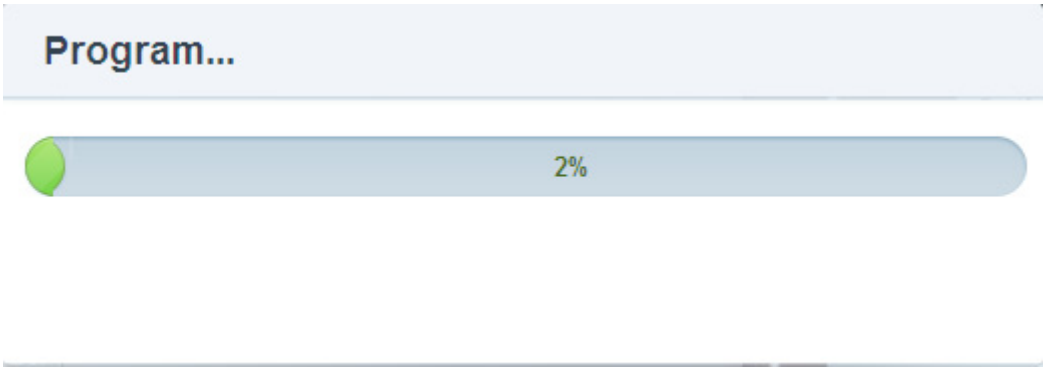


Figure 26

When the program operation is completed, the process bar will display 100%, and a <Close> button will be shown, as shown in Figure 27.

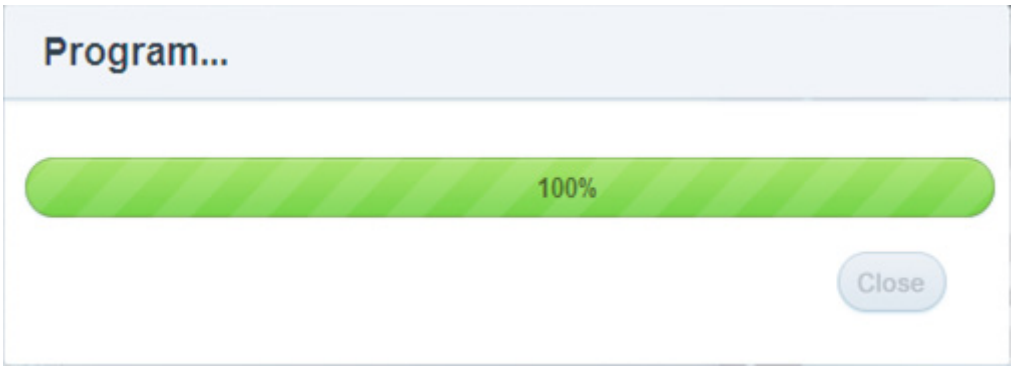


Figure 27

Click the <Close> button to close the process bar thus finishing the program operation.

2 Software Development Kit – SDK

After the I3000 installation is completed, the installation directory includes an ISPDLL folder, which contains two I3000 SDK versions, the x86 used for 32-bit program and the x64 used for 64-bit program. Both of the SDK versions contain all the I3000 functions. Users can choose any one version according to their individual requirements.

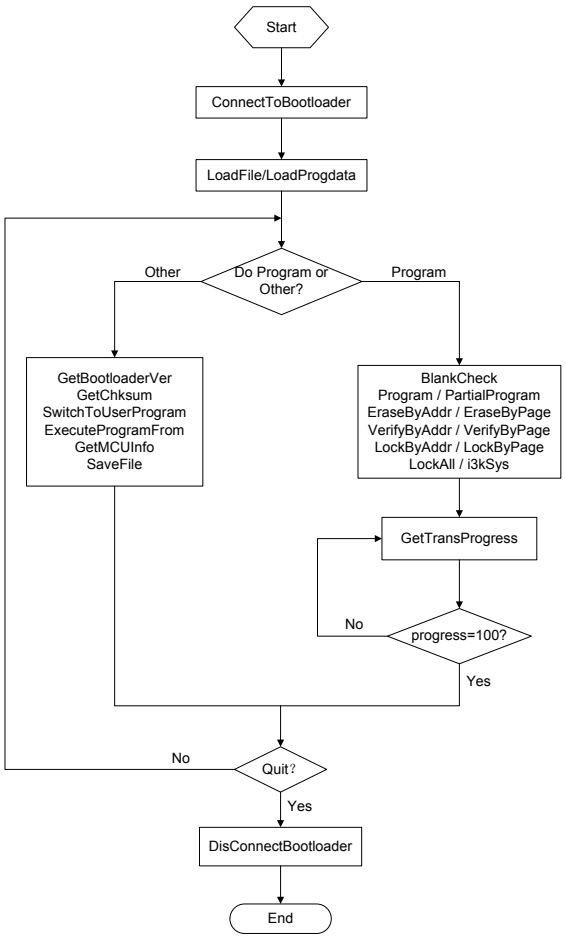
2.1 Operating Introduction

There are two methods to access the DLL file, explicit linking and implicit linking. When using implicit linking, an ISPDLL.h file must be included in the project and linked to the library file ISPDLL.lib. To avoid conflict, users should not define WRITERCMD_API in the user application program.

When the application program is running, the ISPDLL.dll file, HIDDLL.dll file and EFORMAT.ini file must be located in the application program folder. The ISPDLL.dll file includes the required API, and the HIDDLL.dll file provides the underlying interface for the ISPDLL.dll file.

Note: the ISPDLL.dll file uses multi-byte encoding.

2.2 Operating Flowchart



2.3 SDK API Introduction

The ISPDLL.dll file includes 23 functions which will be introduced below.

LoadFile

The function is used to load the MTP file and return the pointers of the Code buffer, Option buffer and Data buffer (if there is any data) and their buffer sizes. Here the buffer sizes are based on the corresponding MCU description in the EFORMAT.ini file.

```
int LoadFile(LPCTSTR szMtpPath, PBYTE& pProgramBuf, WORD& wProgramSize,
PBYTE& pOptionBuf, WORD& wOptionSize, PBYTE& pDataBuf, WORD& wDataSize);
```

Parameters

`[in] szMtpPath`

This is the full path of the MTP file to be loaded. If this parameter is input incorrectly, the function will fail.

`[out] pProgramBuf`

Points to the Code buffer.

`[out] wProgramSize`

Indicates the Code buffer size in words.

`[out] pOptionBuf`

Points to the Option buffer.

`[out] wOptionSize`

Indicates the Option buffer size in words.

`[out] pDataBuf`

Points to the Data buffer, if there is no data, it will return NULL.

`[out] wDataSize`

Indicates the Data buffer size in bytes. If there is no data, it will return 0.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

This function needs to internally load an EFORMAT.ini configuration file. If there is no EFORMAT.ini file in the application directory, the function will fail. Either the LoadFile or LoadProgdata function must be called before a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr, VerifyByPage, LockByAdrr, LockByPage, LockAll or GetChksum.

Sample Code

```
char * pszPath = "C:\\HT68FB550.MTP";
PBYTE pProgram = NULL, pOption = NULL, pData = NULL;
WORD wProgramSize = 0, wOptionSize = 0, wDataSize = 0;
 if (LoadFile(pszPath, pProgram, wProgramSize, pOption, wOptionSize,
pData, wDataSize) != -1)
 {
        //Execute other operations……
}
```

LoadProgdata

Being the same as the LoadFile function, the function uses the loaded MTP file to return the pointers of the Code buffer, Option buffer and Data buffer (if there is any data) and their buffer sizes. Here the buffer sizes are based on the corresponding MCU description in the EFORMAT.ini file.

```
int LoadProgdata(PBYTE pMtpBuf, DWORD dwMtpSize, PBYTE& pProgramBuf,
WORD& wProgramSize, PBYTE& pOptionBuf, WORD& wOptionSize, PBYTE&
pDataBuf, WORD& wDataSize);
```

Parameters

`[in] pMtpBuf`

Points to the MTP buffer but can not be NULL, or the function will fail.

`[in] dwMtpSize`

Indicates the MTP buffer size but can not be 0, or the function will fail.

`[out] pProgramBuf`

Points to the Code buffer.

`[out] wProgramSize`

Indicates the Code buffer size in words.

`[out] pOptionBuf`

Points to the Option buffer.

`[out] wOptionSize`

Indicates the Option buffer size in words.

`[out] pDataBuf`

Points to the Data buffer. If there is no data, it returns NULL.

`[out] wDataSize`

Indicates the Data buffer size in bytes. If there is no data, it returns 0.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

This function needs to load an EFORMAT.ini configuration file internally. If there is no EFORMAT.ini file in the application directory, the function will fail. Either the LoadFile or LoadProgdata function must be called before a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr, VerifyByPage, LockByAddr, LockByPage, LockAll or GetChksum.

Sample Code

```
char * pszPath = "C:\\HT68FB550.MTP";
PBYTE  pMtpBuf = NULL; //MTP file buffer
DWORD dwMtpFileSize = 0; //File size, initialized as 0
ReadFileToBuffer(pszPath, pMtpBuf, dwMtpFileSize); //Read file to buffer
PBYTE pProgram = NULL, pOption = NULL, pData = NULL;
WORD wProgramSize = 0, wOptionSize = 0, wDataSize = 0;
 if (LoadProgdata(pMtpBuf, dwMtpFileSize, pProgram, wProgramSize,
     pOption, wOptionSize, pData, wDataSize) != -1)
 {
        // Execute other operations……
}
```

SaveFile

This function is used to save the MTP file.

```
BOOL SaveFile(LPCTSTR szFilePath);
```

Parameter

`[in] szFilePath`

This is the save path for the MTP file. If this parameter is input incorrectly, the function will fail.

Return Value

If the function succeeds, it returns TRUE; otherwise FALSE.

Note

Ensure that the LoadFile or LoadProgdata function has been called successfully before this function.

Sample Code

```
char * pszPath = "C:\\HT68FB550.MTP";
PBYTE pProgram = NULL, pOption = NULL, pData = NULL;
WORD wProgramSize = 0, wOptionSize = 0, wDataSize = 0;
 if (LoadFile(pszPath, pProgram, wProgramSize, pOption, wOptionSize,
     pData, wDataSize) != -1)
 {
        ASSERT(pProgram != NULL);
        ASSERT(pOption != NULL);
        //Modify Program
        pProgram[wProgramSize/2] = 0x48;
        //Modify Option
        pOption[wOptionSize/2] = 0x59;
        //Modify Data
        if (pData != NULL && wDataSize != 0)
        {
               pData[wDataSize/2] = 0x58;
        }
SaveFile("C:\\HT68FB550_1.MTP");
}
```

GetChksum

This function is used to get the checksum.

```
int GetChksum(BYTE calcMode);
```

Parameter

`[in] calcMode`

Specifies the check type. 0 stands for "Code", 1 stands for "Code + Option", 2 stands for "Code + Option + Data". For other values, the function will fail.

Return Value

If the function fails, the return value is -1, which indicates a wrong check type; otherwise, the return value is a valid checksum.

Note

Ensure that the LoadFile or LoadProgdata function has been called successfully before this function. After the function is called, the program must first check if the return value is -1, if not, then convert the checksum in words or in bytes according to the requirement.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
int nChecksum = GetChksum(0);  //Get Code buffer checksum
if (nChecksum != -1)
{
        printf("checksum=%d\n", nChecksum);  //Print the checksum
}
```

BlankCheck

The function checks whether the specified type of data in the MCU is blank.

```
int BlankCheck(BYTE ucType);
```

Parameter

[in] ucType

Stands for the data type to be checked. 0 stands for Code, 1 stands for Option. For other values, the function will fail.

Return Value

If the function fails, the return value is -1.

If the function succeeds, a return value of 1 stands for non-blank, 0 stands for blank.

Note

The size of the Code or Option buffer is read from the EFORMAT.ini using the MTP file. Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Code buffer BlankCheck, the following code runs in a separate thread
int nRet = BlankCheck(0);
if(0 == nRet)
 puts("Blank!");
else if (1 == nRet)
puts("Non-blank!");
else
puts("BankCheck Failed!");
```

Program

The function programs the specified type of data to the IC.

```
int Program(BYTE ucType);
```

Parameter

`[in] ucType`

Indicates the data type to be programmed. 0 stands for Code, 1 stands for Option. For other values, the function will fail.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

Ensure that the LoadFile or LoadProdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Programming code, runs in a separate thread
int nRet = Program(0);
 if (nRet != -1)
        puts("Program Success!");
 else
        puts("Program Failed!");
```

PartialProgram

The function programs the specified data range to the IC.

```
int PartialProgram(WORD wOffset, WORD wordLen);
```

Parameters

`[in] wOffset`

The starting address of the specified data range, in words. If this address exceeds the maximum address of the MCU available space, the function will fail.

`[in] wordLen`

The total length of data to be programmed, in words. If this length is greater than the MCU available space size, the excess part will be truncated.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

This function is only available for Code programming. The Option programming does not support the function.

This function converts the specified address range into pages. If the address range indicated by wOffset/wordLen is not enough for integer pages, the function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Program 32 words starting from address 0, the following code runs
//in a separate thread
int nRet = PartialProgram(0, 32);
if (nRet != -1)
       puts("PartialProgram Success!");
 else
       puts("PartialProgram Failed!");
```

EraseByAddr

The function erases the specified data by address.

```
int EraseByAddr(WORD wOffset, WORD wordLen, BYTE ucType);
```

Parameters

`[in] wOffset`

The starting address of the address range to be erased, in words. If this parameter is greater than the maximum address of the MCU available space, the function will fail.

`[in] wordLen`

The total length of the address range to be erased, in words. Note that ucType = 0, wordLen can not be 0, or the function will fail. If this parameter exceeds the MCU available space size, the excess part will be truncated.

`[in] ucType`

Indicates the programmed data type. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType=1, the above two parameters will be ignored.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

When ucType=0, this function will convert the specified address range into pages. If the address range indicated by wOffset/wordLen is not enough for integer pages, the function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
///Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Erase 32 words starting from address 0 of the code buffer, the
//following code runs in a separated thread
int nRet = EraseByAddr(0, 32, 0);
 if (nRet != -1)
        puts("Erase Success!");
 else
        puts("Erase Failed!");
```

EraseByPage

The function erases the specified data by page.

```
int EraseByPage(WORD wStartPage, WORD wCnt, BYTE ucType);
```

Parameters

`[in] wStartPage`

Specifies the zero-based index of starting page to be erased. If it is greater than the maximum page number in the MCU, the function will fail.

`[in] wCnt`

The number of pages to be erased. When ucType = 0, this parameter can not be 0, or the function will fail. If this parameter is greater than the maximum number of pages in the MCU, the excess part will be truncated.

`[in] ucType`

Indicates the data type to be erased. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType = 1, the above two parameters will be ignored.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Erase 100 pages starting from page 0 of the code buffer, the
//following code runs in a separated thread
int nRet = EraseByPage(0, 100, 0);
 if (nRet != -1)
        puts("Erase Success!");
 else
        puts("Erase Failed!");
```

VerifyByAddr

The function verifies whether the data of the specified addresses in the MCU match the programmed data.

```
int VerifyByAddr(WORD wOffset, WORD wordLen, BYTE ucType);
```

Parameters

`[in] wOffset`

The starting address of the specified address range to be verified, in words. If this parameter is greater than the maximum address of the MCU available space, the function will fail.

`[in] wordLen`

The total length of the addresses to be verified, in words. When ucType = 0, wordLen can not be 0, or the function will fail. If this parameter exceeds the MCU available space size, the excess part will be truncated.

`[in] ucType`

The data type to be verified. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType = 1, the above two parameters will be ignored.

Return Value

If the function fails, the return value is -1.
If the function succeeds, a return value of 0 indicates that all data matched, 1 indicates unmatched.

Note

When ucType =0, this function will convert the specified address range into pages. If the address range indicated by wOffset/wordLen is not enough for integer pages, the function will internally align it into integer pages. If the address range exceeds the available space, the excess part will be truncated.

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Verify 64 words starting from address 0 of the code buffer, the
//following code runs in a separated thread
int nRet = VerifyByAddr(0, 64, 0);
 if (0 == nRet)
       puts("Data Matched!");
 else if (1 == nRet)
       puts("Data Unmatched!");
 else
       puts("Function fails!");
```

VerifyByPage

The function verifies whether the data of the specified pages in the MCU match the programmed data.

```
int VerifyByPage(WORD wStartPage, WORD wCnt, BYTE ucType);
```

Parameters

`[in] wStartPage`

Specifies the zero-based index of the starting page to be erased. If it is greater than the maximum page number in the MCU, the function will fail.

`[in] wCnt`

The number of pages to be verified. When ucType = 0, this parameter can not be 0, or the function will fail. If this parameter is greater than the maximum number of pages in the MCU, the excess part will be truncated.

`[in] ucType`

Indicates the data type to be verified. 0 stands for Code, 1 stands for Option. For other values, the function will fail. When ucType = 1, the above two parameters will be ignored.

Return Value

If the function fails, the return value is -1.
If the function succeeds, a return value of 0 indicates that all data is matched, 1 indicates unmatched.

Note

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Verify 100 pages starting from page 0 of the code buffer, the
//following code runs in a separated thread
int nRet = VerifyByPage(0, 100, 0);
if (0 == nRet)
       puts("Data Matched!");
 else if (1 == nRet)
       puts("Data Unmatched!");
 else
       puts("Function fails!");
```

LockByAddr

The function locks pages where the specified addresses exit.

```
int LockByAddr(WORD wOffset, WORD wordLen);
```

Parameters

`[in] wOffset`

The starting address of the specified addresses to be locked, in words. If this parameter is greater than the maximum address of the MCU available space, the function will return 1.

`[in] wordLen`

The total length of addresses to be locked, in words. If this parameter is greater than the MCU available space size, the function will return 1.

Return Value

If the function fails, the return value is -1 which stands for a hardware error.
If the function succeeds, a return value of 1 indicates wrong pages, 0 indicates a successful lock.

Note

This function converts the address range specified by wOffset/wordLen into pages, 256 words as one page. If the specified address range is not enough for integer pages, this function will internally align it into integer pages. The Lock function usually locks two pages at once. If the required pages number is odd, the actual locked pages number is even.

If the address range specified by wOffset/wordLen is equal to the MCU Code buffer size, then the whole Code buffer will be locked (Program Rom). Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Lock 320 words starting from address 0
int nRet = LockByAddr(0, 320);
 if (0 == nRet)
   puts("Lock Success!");
 else if (1 == nRet)
   puts("Wrong Pages!");
else
   puts("Lock Failed!");
```

LockByPage

This function locks the specified pages.

```
int LockByPage(WORD wStartPage, WORD wCnt);
```

Parameters

`[in] wStartPage`

Specifies the zero-based index of starting page to be locked. If it is greater than the maximum page number to be locked, the function will fail.

`[in] wCnt`

The number of pages to be locked. If this parameter is greater than the maximum number of pages in the MCU, the excess part will be truncated.

Return Value

If the function fails, a return value of 1 indicates wrong pages, -1 indicates a hardware error.
If the function succeeds, the return value is 0 indicating a successful lock.

Note

If the page range specified by wStartPage/wCnt is equal to the MCU Code buffer size, then the whole Code buffer will be locked (Program Rom). Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
//Lock 5 pages starting from page0: page0, page1, page2, page3 and page4
int nRet = LockByPage(0, 5);
if (0 == nRet)
  puts("Lock Success !");
else if (1 == nRet)
  puts("Wrong Pages!");
else
  puts("Lock Failed!");
```

LockFromOption

This function locks pages according to the option.

```
int LockFromOption (PBYTE pOption, WORD wordLen);
```

Parameters

`[in] pOption`

The address that points to the option buffer.

`[in] wordLen`

Option length.

Return Value

If any one of the above parameters is invalid, the return value is 1.
If the function succeeds, the return value is 0, otherwise -1.

Note

This is a new and higher-level function which is added from V2.2.5.14. In general cases, users should not call this function directly but call the LockByAddr, LoadByPage or LoadAll function. Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
BYTE byOption[64] = {…};
int nRet = LockFromOption (byOption, sizeof(byOption)/2);
if (0 == iRet)
   puts("Execute Success!");
else if (1 == iRet)
   puts("Wrong Parameter!");
else
   puts("Execute Failed!");
```

LockAll

This function locks the whole MCU.

```
BOOL LockAll();
```

Return Value

If the function succeeds, it returns TRUE; otherwise FALSE.

Note

Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

GetTransProgress

The function gets the current programming progress.

```
int GetTransProgress();
```

Return Value

The function returns the current completion ratio.

Note

After a program type function, such as BlankCheck, Program, PartialProgram, EraseByAddr, EraseByPage, VerifyByAddr or VerifyByPage, is called, this function can be called to get the related programming progress, a completion ratio of 100 indicates that the program operation is finished.

Sample Code

```
int nProgress = GetTransProgress();
printf("Curret Progress:%d%%", nProgress);
```

GetBootloaderVer

The function gets the Bootloader version.

```
BYTE GetBootloaderVer();
```

Return Value

The Bootloader version.

Sample Code

```
//Ensure that the ConnectToBootloader function has been called successfully,
//here omit the detailed procedure
BYTE version = GetBootloaderVer();
printf("Bootloader version: %d", version);
```

ConnectToBootloader

The function is used to connect to the Bootloader.

```
int ConnectToBootloader();
```

Return Value

If the function fails, the return value is -1; otherwise 0.

Sample Code

```
if (ConnectToBootloader() != -1)
    puts("Connection Successful!");
else
    puts("Connection Failed!");
```

DisConnectBootloader

The function is used to disconnect the Bootloader.

```
int DisConnectBootloader();
```

Return Value

If the function fails, the return value is -1; otherwise 0.

Sample Code

```
if (DisConnectBootloader() != -1)
puts("Disconnection Sucessful!");
else
puts("Disconnection Failed!");
```

SwitchToUserProgram

The function is used to switch to the user program.

```
void SwitchToUserProgram();
```

Note

This function usually works together with the ExecuteProgramFrom function.

Sample Code

```
if (ExecuteProgramFrom(0) == 0)
    SwitchToUserProgram();
```

ExecuteProgramFrom

The function is used to setup the program executing mode.

```
int ExecuteProgramFrom(BYTE byMode);
```

Parameter

`[in] byMode`

Indicates the program executing mode.

0: MCU Starts at the bootloader;

1: MCU starts at the bootloader then executes the user firmware;

2: MCU starts at the user firmware;

For other values, the function will fail.

Return Value

If the function fails, the return value is -1; otherwise 0.

Note

This function is usually called before the SwitchToUserProgram function.

Sample Code

```
if (ExecuteProgramFrom(0) == 0)
    SwitchToUserProgram();
```

GetMCUInfo

The function gets the MCU information.

```
BOOL GetMCUInfo(MCUINFO* pMcuInfo);
```

Parameter

[in/out] pMcuInfo

Points to the MCUINFO structure which is defined in ISPDLL.h.

```
typedef struct MCUINFO_t
{
    DWORD cbSize;        //structure size
    int nPageSize;       //page size for the programming operation, in words
    int nMaxProgramPage; //maximum page number that can be programmed
    int nMaxLockPage;    //maximum page number that can be locked
    int nBootloaderSize; //Bootloader size, in words
    char szMcuName[16];  //the current MCU name, a new variable which is
                         //added in v2.2.5.14
}MCUINFO;
```

Return Value

If the function succeeds, it returns TRUE; otherwise FALSE.

Note

When passing the parameter, user must first assign to cbSize using sizeof(MCUINFO). Ensure that the LoadFile or LoadProgdata function has already been called successfully before this function.

The nMaxLockPage is a member of the MCUINFO structure indicating the whole Program Rom size, the value of which is irrelevant to the bootloader. The Program Rom whole size minus the nBootloaderSize value makes the nMaxProgramPage value. Here nBootloaderSize is assigned based on the bootloader description in the MTP file. If the MTP file includes no bootloader description, then nBootloaderSize equals to 0.

Sample Code

```
//Ensure that the LoadFile or LoadProgdata function has been called
//successfully, here omit the detailed procedure
MCUINFO mi;
mi.cbSize = sizeof(MCUINFO);
if (GetMCUINFO(&mi))
{
 printf("Current MCU:%s, PageSize:%d, MaxProgramPage:%d,
 MaxLockPage:%d, Bootloader Size:%d",
 mi.szMcuName, mi.nPageSize, mi.nMaxProgramPage, mi.nMaxLockPage,
 mi.nBootloaderSize);
}
```

i3kSys

The function reads a given size of data from MCU Code buffer and compares them with the specified data.

```
BOOL i3kSys(WORD wOffset, WORD wSize, BYTE* buffer);
```

Parameter

`[in] wOffset`

The starting address of the MCU Code data to be compared, in words.

`[in] wSize`

The size of the MCU Code data to be compared, in words. If this parameter is 0 or not an integer multiple of the page size, the function will return FALSE.

`[in] buffer`

Points to the specified data buffer.

Return Value

If the data read from the Code buffer match the specified data, the function will return TRUE, otherwise FALSE. In addition, if any one of the above parameters is invalid, the function will also return FALSE.

Sample Code

```
// Ensure that the LoadFile or LoadProgdata function has been called
// successfully, here omit the detailed procedure.
// First to get the pointer of the specified buffer to be compared
// Then call GetMCUInfo(1)
// Compare n pages starting from address 0 of the Code space with the
// specified buffer (here n is a integer number except 0).
if (i3kSys(0, GetMCUInfo(1)*n, buffer) != FALSE)
        puts("TRUE");
else
        puts("FALSE");
```

Copyright© 2014 by HOLTEK SEMICONDUCTOR INC.

The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.tw.